|  |  |
| --- | --- |
|  |  |
| 项目报告B |  |
|  |  |
|  | 课程：数据挖掘姓名：姜其升 |
|  | 学号：1752058指导教师：饶卫雄教授 |

[项目报告B 1](#_Toc39129537)

[1 算法概述 3](#_Toc39129542)

[1.1 Jaccard系数计算 3](#_Toc39129543)

[1.2 簇的质心计算 3](#_Toc39129544)

[1.3 聚类过程 3](#_Toc39129545)

[1.3.1 选择初始质心 3](#_Toc39129546)

[1.3.2 进行聚类 3](#_Toc39129547)

[1.3.3 重新分配初始点 3](#_Toc39129548)

[1.4 聚类结果评价 3](#_Toc39129549)

[1.5 选择正确的K值 4](#_Toc39129550)

[2 关键代码 5](#_Toc39129551)

[3 聚类结果及运行截图 6](#_Toc39129552)

[4 聚类结果分析 9](#_Toc39129553)

[4.1 距离分布情况 9](#_Toc39129554)

[4.2 SC/CP曲线图 9](#_Toc39129555)

[5 算法性能 11](#_Toc39129556)

## 1 算法概述

### 1.1 Jaccard系数计算

分别计算每位顾客在第一级品类结构到第四级品类结构购买不同种商品的总价，两位顾客共同购买相同（交集）商品的金额总额与二者购买商品的并集的商品金额总额之比，sim1，sim2，sim3，sim4。Jaccard系数为四个sim的平均值。

Jaccard系数越接近1，则说明二者越相似，反正则差异越大。

### 1.2 簇的质心计算

本题中采用簇内用户购买各种级别的品类结构的各个品类商品总额的平均值作为簇的质心。

### 1.3 聚类过程

本题使用K-means聚类方式，具体步骤如下。

#### 1.3.1 选择初始质心

按照给定的期望簇的个数K，寻找初始质心。采用寻找到的方式为：选择彼此距离尽可能远的那些点（即最不相似的点，Jaccard越接近0 的点）。具体做法如下：

1. 随机选择第一个点;
2. WHILE 已选择的质心数 < K :
3. 选择离已经选择的所有质心中距离最远的点加入到质心中;
4. END;

#### 1.3.2 进行聚类

这是算法的核心部分，我们考虑将剩余的点分配给最近的簇（离质心最近）。需要注意，当点分配到簇后，簇的质心可能会改变，因此，每次我们需要更新质心。具体步骤如下：

1. FOR 每个剩余的点p DO:
2. 找到离点p最近的质心;
3. 把点p加入到这个质心所在的簇中;
4. 调整这个簇的质心;
5. END;

#### 1.3.3 重新分配初始点

聚类的最后一部，我们固定所有簇的质心，然后将包括K个初始值的所有点重新分配到这K个簇中。

### 1.4 聚类结果评价

我们这里采用Silhouette Coefficient（SC）和Compactness（CP）进行评价。

计算时，样本间距离用Jaccard距离（1 – Jaccard系数）进行计算。

其中a(i)为样本i与同一簇内其他样本的平均距离，b(i)为样本i与不同簇的其他所有样本的平均距离。SC描述了簇的内聚情况，与其他簇的边界是否清晰，越接近与1，效果越好。

其中ct为每个簇的质心，CP描述了簇每个成员距离质心的平均距离，一个较小的CP值通常意味着更好的聚类效果。

### 1.5 选择正确的K值

我们这里采用不同的K值进行聚类，之后比较SC和CP两个指标进行聚类的质量评价。首先对K=1, 2, 4, 8, 16, … 运行聚类算法，最终我们可以得到两个K值v和2v，这两次聚类之间的指标差值很低，那么我们可以知道数据中的K值子啊v/2和v之间，我们在这个范围内挑选一个较为合适的值，作为最终的K值，得到最后的 聚类算法。

## 2 关键代码

1. # 数据预处理
2. csv\_data = pd.read\_csv('trade\_new.csv')
3. csv\_data['type4'] = csv\_data.apply(**lambda** x: get4(x['pluno']), axis=1)
4. memberSum = csv\_data.groupby([csv\_data['vipno'], csv\_data['type4']])['amt'].sum()
5. # 计算Jaccard系数矩阵
6. matrix = np.matlib.ones((size, size))
7. **for** i **in** range(size):
8. **for** j **in** range(i+1, size):
9. matrix[i,j] = jaccard(customer[i], customer[j])
10. matrix[j,i] = matrix[i,j]
12. # 初始化质心，设置种子方便复现
13. random.seed(2020)
14. beginCentroids = [int(size \* random.random())]
15. **while** len(beginCentroids) < pointNum:
16. minIndex = 0
17. minValue = 1
18. **for** i **in** range(size):
19. beginCentroids.append(minIndex)
20. # 开始聚类
21. **for** i **in** beginCentroids:
22. clusters.append([customer[i]])
23. nowCentroids.append(memberSum.loc[customer[i]])
24. # 其他点聚类
25. **for** i **in** range(size):
26. nowCentroids[clusterIndex] = nowCentroids[clusterIndex].apply(**lambda** x:x/len(clusters[clusterIndex]))
27. # 起始点聚类
28. **for** i **in** range(pointNum):
29. nowCentroids[i] = nowCentroids[i].apply(**lambda** x:0 **if** len(clusters[i])==0 **else** x/len(clusters[i]))
30. nowCentroids[clusterIndex] = nowCentroids[clusterIndex].apply(**lambda** x:0 **if** len(clusters[clusterIndex])==0 **else** x/len(clusters[clusterIndex]))
31. # 计算sc
32. sc = 0
33. **for** i **in** range(pointNum):
34. sc = sc + (b-a)/max(a, b)
35. sc\_list.append(sc/size)
36. # 计算cp
37. cp = 0
38. **for** i **in** range(pointNum):
39. cp = cp + cpi/len(clusters[i])
40. cp\_list.append(cp/pointNum)

## 3 聚类结果及运行截图

K = 16

SC = 0.063

CP = 0.669

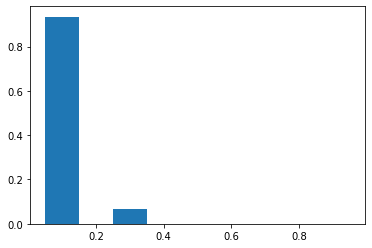
具体结果：

[2900000383913, 13325038116, 13854627199, 15954688237, 15963883482, 18654692914, 1590120464497, 1590130640102, 1590130817948, 1590140304506, 1590140305107, 1590140306678, 1590140307286, 1590140307415, 1590140307767, 1590140307859, 1590140308030, 1590140308214, 1590142128124, 1590142129763, 1590142148962, 1590142149853, 1590142151689, 1590142179621, 1590142191722, 1590142194327, 1590142197076, 1590142201162, 1590142203104, 1590142203180, 1590142203401, 1590142205993, 1590142206747, 1590142210195, 1590142213189, 1590142213356, 1590142213981, 1590142215619, 1590142242370, 1590142477895, 1590142519632, 1590142520300, 1590142648356, 1590142751995, 1590151076256, 1590151321240, 1590151363066, 1590151464053, 1590151472027, 1590151825175, 1590151832586, 1590151832593, 1590151835808, 1590151843124, 1591013471721, 1591013766650, 1591013866800, 1591013877134, 1591014338276, 1591014523610, 1591014680184, 1591015097233, 1591015269418, 1591015327590, 1591015408602, 1591015419448, 1591015452933, 1591015454210, 1591015454562, 1591015454814, 1591015456955, 1591015476700, 1591015478124, 1591015480745, 1591015483951, 1591015519216, 1591016154157, 1591016162466, 1591016170300, 1591016174353, 1591016174957, 1591016411137, 1591016440328, 1591016517778, 1591016519031, 1591020742401, 1591030217500, 1591030411823, 1591040087452, 1591040416658, 1591040462983, 1592010017509, 1592014016171, 1592015013285, 1592015018662, 1592015046818, 1592015053267, 1592015056299, 1592015059993, 1593141148827, 1593150463515, 1594140460286, 1595140489499, 1595150403423, 1595150770945, 1595150771652, 1595151204906, 1595151442803, 1595151616471, 1595151629143, 1595151630446, 1595151630774, 1595151866807, 1595151915963, 1595160035249, 1595160353664, 1596140586607, 1596140587802, 1596160006376, 1596160024097, 1598140107578, 2900000040663, 2900000072022, 2900000074163, 2900000124974, 2900000125261, 2900000138056, 2900000164567, 2900000179417, 2900000203457, 2900000239876, 2900000245396, 2900000245914, 2900000246638, 2900000268692, 2900000308312, 2900000337633, 2900000342651, 2900000345300, 2900000350175, 2900000350458, 2900000353848, 2900000372849, 2900000394940, 2900000404731, 2900000416307, 2900000418073, 2900000448520, 2900000449732, 2900000452756, 2900000476042, 2900000485990, 2900000557161, 2900000562165, 2900000566378, 2900000587649, 2900000604360, 2900000781603, 2900000840164, 2900000840805, 2900000847224, 2900000849785, 2900000863880, 2900000874145, 2900000874930, 2900000884007, 2900000890688, 2900000934474, 2900000936997, 2900000941830, 2900001050500, 2900001050791, 2900001067911, 2900001094702, 2900001211239, 2900001211321, 2900001215688, 2900001220255, 2900001234399, 2900001240246, 2900001270199, 2900001307901, 2900001308168, 2900001421652, 2900001432658, 2900001436700, 2900001463379, 2900001465960, 2900001467353, 2900001497336, 2900001509459, 2900001540117, 2900001550246, 2900001568630, 2900001575201, 2900001578363, 2900001661508, 2900001663823, 2900001671408, 2900002927733, 2900002930092, 2900002932614, 2900002933024, 2900002933918, 2900002934304, 2900002934892, 2900002936520, 2900002936940, 2900002937428, 2900002969559, 2900003106281, 2900003106502, 2900003110479, 2900003123479, 6222021615010082737, 6227002180901670266]  
[2900001053938, 1595150738976, 2900001466509, 2900002532630]  
[1591015091286]  
[1591015454142, 1590151829753, 2900000060012, 2900000856585]  
[1595151786686, 1591015480455, 1591015598785, 1591016350504, 1594140121125, 2900000047648, 2900000384187, 2900000401495, 2900000468030, 2900001137843, 2900001355049]  
[1590142240482, 1590142201513, 1590142202190, 1590142516600, 1590151457185, 1590151459721, 1590151682891, 1591014943968, 1591015027858, 1591015088262, 1592015026841, 2900000307377, 2900000516892, 2900001413886, 2900001436120, 2900002542837, 6227002180921663895]  
[1595150997786, 1591015501716, 1592015016880, 1592015058781, 1595150263683, 2900000330863, 2900000408586, 2900001157902, 2900002937503]  
[2900000574274, 18764569988, 1591015034924, 1595151355660, 1596130424193, 2900000776104, 2900001470780, 2900003105161]  
[1590151977348, 1590142149112, 1590142201148, 1590142518307, 1590151103907, 1590151427355, 1590151428581, 1591012978559, 1591014083046, 1591014214433, 1591015097516, 1591015184407, 1591015218485, 1591015420000, 1591015454692, 1591015521639, 1591015600815, 1591015604417, 1591016172045, 1591016310287, 1591016517921, 1591020377344, 1592013087615, 1595151631146, 1596140594374, 1596160058337, 2900000050785, 2900000071742, 2900000072701, 2900000177079, 2900000178106, 2900000224001, 2900000255463, 2900000256637, 2900000271814, 2900000284333, 2900000304376, 2900000386440, 2900000386495, 2900000390256, 2900000451452, 2900000490130, 2900000588691, 2900000623040, 2900000784918, 2900000845121, 2900000852525, 2900000863286, 2900000864498, 2900000883475, 2900001047739, 2900001050364, 2900001067881, 2900001068499, 2900001068833, 2900001090605, 2900001093019, 2900001095952, 2900001216616, 2900001222099, 2900001238632, 2900001255080, 2900001270205, 2900001331616, 2900001364058, 2900001398121, 2900001419826, 2900001431651, 2900001451932, 2900001452366, 2900001465380, 2900001465816, 2900001559720, 2900001645768, 2900001668996, 2900002548457, 2900002701746, 2900002932416, 2900002934342, 2900002934915, 2900002944471, 2900002944495, 2900003105154, 2900003107363, 2900003108773, 2900003114002, 2900003114170, 2900003115009, 2900003120607]  
[1595151375125, 1591015506575, 1592015056343, 1595151609213, 2900000210790, 2900000317307, 2900000339637, 2900000780835, 2900000883819, 2900001078658, 2900001270014, 2900001436366, 2900001437165, 2900001540971, 2900001582933, 2900002934762]  
[1592015058774, 1590120718170, 1590142517836, 1591013227632, 1591040203906, 1595151644528, 2900000048812]  
[1591020742449, 1591014130276, 1591040356657, 1595130081566, 2900000224315, 2900000348561, 2900000417663, 2900000780040, 2900000855502, 2900003117386]  
[2900000254503, 781924, 13864739266, 15963885355, 18554652702, 1590141216259, 1590141414228, 1590142152440, 1590142456470, 1590142514866, 1590150321449, 1590151500423, 1591013167365, 1591014278527, 1591014637324, 1591015074944, 1591015136338, 1591015456993, 1591016159916, 1591016320026, 1591016382857, 1591016443121, 1591016494765, 1591020695158, 1591030092053, 1591040155274, 1592015054592, 1593160622155, 1595150992095, 1595151168727, 1595151738968, 1598140129341, 2900000194298, 2900000216952, 2900000680036, 2900000780743, 2900001109598, 2900001309639, 2900001423069, 2900001435116, 2900001436410, 2900001437424, 2900001538596, 2900001684507, 2900002944402, 2900003114880, 2900003116822]  
[1591040497398, 15954611837, 1590140304209, 1590142150507, 1590142176354, 1590142204569, 1590142230179, 1590142516068, 1590151210391, 1590151432557, 1590151467962, 1591014133086, 1591015112424, 1591015457273, 1591015500993, 1591016439575, 1591040161114, 1593140971419, 1595112593216, 1595140612422, 1595150738747, 1595151123344, 1596140628703, 1598140055923, 2900000050419, 2900000138223, 2900000253230, 2900000385627, 2900000880467, 2900001020541, 2900003114613, 2900003116341]  
[2900000870994, 1590142176521, 1590151472782, 1590151842899, 1591014255948, 1591014577972, 1591020269281, 1595150883423, 2900000192492, 2900000544765, 2900001092012, 2900001432672, 2900001462648, 2900003122076]  
[2900000598973]



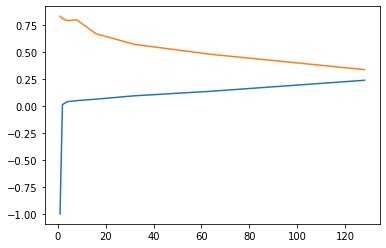
## 4 聚类结果分析

### 4.1 距离分布情况



上图为Jaccard系数的分布情况，我们可以看到，分布较为集中，超过90%的顾客间相似系数分布在0~0.2这个区间之内，剩余不到10%则主要分布于0.2~0.4区间，其余部分几乎没有分布，分布较为集中，按照此方法计算顾客相似度较小，分布过于集中，不利于聚类，聚类结果可能较差，区分度不大。

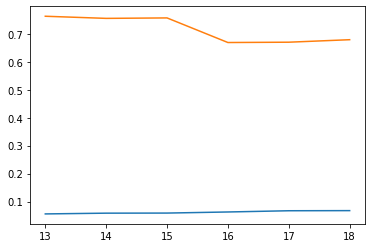
### 4.2 SC/CP曲线图



首先我们看到的是数据从[1, 2, 4, 8, 16, 32, 64, 128]变化的情况图，我们可以看到蓝色曲线为SC的值随K变化的情况，橙色曲线为CP的值随K值变化的情况。

首先我们明确一下几点，当K为1时，所有的点都在一个簇内，在SC公式中b(i)为0，因此SC为-1；当K最大，即为顾客的个数（一个顾客为一簇），此时在SC公式中a(i)为0，此时SC为1，最大；当K最大时，在CP公式中，每个距离为0，因此CP为0，最小。

图中趋势符合上述分析。同时，由以上分析可知，一味追求SC最大或CP最小时不可取的，因此我们采用在2中提到的方式确定最适宜的K值。SC值变化并不明显，但观察CP曲线，我们可以看到在16之后的CP的值的减少幅度明显小于16之前的变化幅度，因而，我们将范围缩短至[13, 14, 15, 16, 17, 18]，由聚类结果可得下图。



从图中我们可以看到更加细微的变化，从15到16CP下降明显，均好于之前和之后的减少幅度，拐点较为明显，因此我们最终决定将K设定为16。最终结果如3中所示。

## 5 算法性能

算法性能较差，时间主要花费在多次重复聚类以及Jaccard系数矩阵的运算上，且随着K的数量的不断增大，速度会更慢。尤其是Jaccard系数矩阵的计算，由于按照四级品类分别计算，因此相比于第一题的计算会重复四次，之后求和，时间明显长于A题中的计算时长。

同时，与存储的数据结构有关，顾客、商品种类及花费钱数采用Pandas.Series存储，还涉及二级索引、多次筛选求和，如下图所示，因此在求Jaccard系数时，需要采用新建Series的方式求交集和并集，效率较低，最终导致耗时较长。如果采用更为方便的存储方式，如顾客与商品的二维金额矩阵，则可以直接使用每个商品的最大值和最小值相加求解，可以加速系数的求解。

